Propositional Resolution

Second Part: Algorithms

Benjamin Wack

Université Grenoble Alpes

January 2025

Proof by resolution of our running example

- (H1) : $p \Rightarrow \neg j \equiv \neg p \vee \neg j$
- (H2) : $\neg p \Rightarrow j \equiv p \vee j$
- (H3) : $j \Rightarrow m \equiv \neg j \vee m$
- ($\neg$ C): $\neg m \wedge \neg p$

Clauses: $\{\neg p \vee \neg j,\ p \vee j,\ \neg j \vee m,\ \neg m,\ \neg p\}$

$$\dfrac{\dfrac{\dfrac{p \vee j \qquad \neg j \vee m}{p \vee m} \qquad \neg m}{p} \qquad \neg p}{\bot} \quad \text{OR} \quad \dfrac{\dfrac{\dfrac{p \vee j \qquad \neg p}{j} \qquad \neg j \vee m}{m} \qquad \neg m}{\bot}$$

OR...

Last course

- Boolean Algebra
- Boolean functions
- Resolution

(1) $A \vdash B$

B is *deduced* from *A*: there is a proof by resolution of *B* starting from *A*.

(2) $A \models B$

B is a *consequence* of *A*: every model of *A* is also a model of *B*.

Today: Correctness

(1) $\Rightarrow$ (2)

Today: Completeness

(2) $\Rightarrow$ (1)

Overview

Correctness

Completeness

Introduction to resolution algorithms

The Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

Complete strategy

Conclusion

Definition

The correctness of a deductive system states that all proofs obtained in this system “prove only true statements”.

Correctness of the resolution rule

Theorem 2.1.15

If C is a resolvent of A and B then $A, B \models C$.

Proof.

If C is a resolvent of A and B, then there is a literal L such that $A = A' + L$, $B = B' + L^c$, and $C = A' + B'$.

Let v be an assignment such that $[A]_v = 1$ and $[B]_v = 1$: let us show that $[C]_v = 1$.

- If $[L]_v = 1$. Then $[L^c]_v = 0$.
  Since $[B]_v = 1$, v is a model of B'. Hence $[C]_v = 1$.
- If $[L^c]_v = 1$. Then $[L]_v = 0$.
  Since $[A]_v = 1$, v is a model of A'. Hence $[C]_v = 1$.

In every case v is a model of C.

□

Correctness of deduction

Theorem 2.1.16

Let Γ be a set of clauses and C a clause. If $\Gamma \vdash C$ then $\Gamma \models C$.

Proof.

Suppose that there is a proof P of C starting from Γ.
Suppose that for any proof of $\Gamma \vdash D$ shorter than P, we have $\Gamma \models D$.
Let us show that $\Gamma \models C$. There are two possible cases:

1. C is a hypothesis in Γ, in this case $\Gamma \models C$.
2. $\Gamma \vdash A$ and $\Gamma \vdash B$ (with a shorter proof) and

$$\frac{A \qquad B}{C}$$

By induction hypothesis: $\Gamma \models A$ and $\Gamma \models B$.
By correctness of the resolution rule: $A, B \models C$. Hence $\Gamma \models C$.

□

Definition

Completeness for refutation: If $\Gamma \models \bot$ then $\Gamma \vdash \bot$.

$\Gamma[L := 1]$

Definition 2.1.18

Let Γ be a set of clauses and L a literal.

$\Gamma[L := 1]$ is obtained by:

- deleting the clauses containing L
- removing L^c from the other clauses.

$\Gamma[L := 0]$ is similarly defined by switching the roles of L and L^c.

Remark: the number of variables in Γ has been decreased.

Examples

Example 2.1.19

Let Γ be the set of clauses $\overline{p}+q$, $\overline{q}+r$, $p+q$, $p+r$. We have:

- $\Gamma[p := 1] =$

  $\{q, \overline{q}+r\}$.

- $\Gamma[p := 0] =$

  $\{\overline{q}+r, q, r\}$.

Notice that:

- $(\overline{1}+q)(\overline{q}+r)(1+q)(1+r) \equiv q(\overline{q}+r)$
- $(\overline{0}+q)(\overline{q}+r)(0+q)(0+r) \equiv (\overline{q}+r)qr$

Property of $\Gamma[L := \ldots]$

Property 2.1.21

Γ has a model if and only if $\Gamma[L := 1]$ or $\Gamma[L := 0]$ has a model.

Proof.

$\Rightarrow$ If v is a model of Γ then v is a model of
either $\Gamma[L := 0]$ (if $[L]_{v'} = 0$)
or $\Gamma[L := 1]$ (if $[L]_{v'} = 1$)

$\Leftarrow$ If v is a model of $\Gamma[L := i]$
then we can build a model of Γ (by taking $[L]_{v'} = i$)

□

Lemma 2.1.22

Lemma 2.1.22

Let Γ a set of clauses, C a clause and L a literal.
If $\Gamma[L := 1] \vdash C$ then $\Gamma \vdash C$ or $\Gamma \vdash C + L^c$.

Proof.

Idea: we put back L^c in the clauses where it was removed.

- If $C \in \Gamma[L := 1]$:
  - either C was in Γ, thus $\Gamma \vdash C$
  - or C was obtained by removing a L^c, thus $\Gamma \vdash C + L^c$
- If C is a resolvent of A and B:
  - either $\Gamma \vdash A$ and $\Gamma \vdash B$, hence $\Gamma \vdash C$
  - or L^c has to be put back into A or B, thus into C too

□

Completeness of propositional resolution

Theorem 2.1.24

Let Γ be a finite set of clauses. If $\Gamma \models \bot$ then $\Gamma \vdash \bot$.

Proof

By induction on the number of variables in Γ.

- Base case: Γ has no variable, so $\Gamma = \emptyset$ (impossible, since $\emptyset$ is valid) or $\Gamma = \{\bot\}$.
- Inductive step: either we prove directly that $\Gamma \vdash \bot$, or that $\Gamma \vdash x$ and $\Gamma \vdash \overline{x}$.

Corollary 2.1.25

Γ is unsatisfiable if and only if $\Gamma \vdash \bot$.

Presentation of two algorithms

How to "systematically" decide whether Γ is inconsistent or not?

- The Davis-Putnam-Logemann-Loveland Algorithm
  "Intelligent" traversal of the possible assignments of Γ
- Complete strategy
  Construction of ALL the deductible clauses (resolvents) from Γ

Remark

Exponential solutions in time in the worst case.

Exponential complexity

Remember that:

- in a clause, the order and repetition of literals are irrelevant ;
- a valid clause (containing p and $\bar{p}$) is useless ;
- in a proof, the same clause can be re-used at will.

How many distinct clauses can you write with n variables?

For each variable p, a clause may contain either:

- p
- $\bar{p}$
- or neither of them

hence 3^n distinct clauses.

Thus, if Γ uses n literals, no proof by resolution may be longer than 3^n lines.

Reduction of a set of clauses

In order to accelerate the algorithms, we reduce the set of clauses.

How to proceed with reduction?

Remove the valid clauses and the clauses containing another clause of the set.

Example 2.1.27

The reduction of the set of clauses
$\{p+q+\overline{p},\ p+r,\ p+r+\overline{s},\ r+q\}$ gives the reduced set:

$\{p+r,\ r+q\}$.

Justification

Property 2.1.28

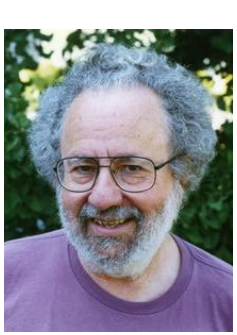
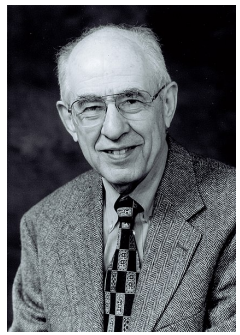
A set of clauses E is equivalent to the reduced set of clauses obtained from E.

Proof.

- Removing valid clauses: $x.1 \equiv x$
- Removing a clause including another clause: $x(x+y) \equiv x$

□

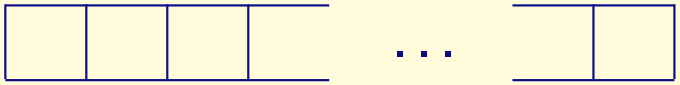
History

- Martin Davis (1928-2023), american mathematician
- Hilary Putnam (1926-2016), american philosopher, mathematician and computer scientist
- **resolution** rule (exhaustively used in the initial algorithm)
- Algorithm for satisfiability of boolean formulas (1960)
  - finds (if possible) a model of a set of clauses
  - initially devised to study first-order formulas
  - refined in 1962 by M. Davis, G. Logemann and D. Loveland
  - basis for efficient SAT-solvers
- Proof of undecidability of Diophantine equations (with Y. Matiyasevich and J. Robinson)

SAT solver demo

Problem



- Each square may either contain a token or not.
- Two neighbouring squares can never both contain a token.
- At least two squares must contain a token.

Boolean modelization

The variable i is true if the square i contains a token.

Resolution using a SAT solver

- Build the clauses
- Translate them into DIMACS format
- Execute and interpret the result

Simplification principles (Davis & Putnam)

Two types of formulas transformations:

1. preserving the truth value:
   - reduction
2. preserving only satisfiability:
   - pure literal elimination
   - unit resolution

DPLL is (usually) efficient because it uses these two kinds transformations.

Splitting principle (Logemann & Loveland)

"Branching/Backtracking"

- **Branching**: After simplification, assign to **true** a heuristically chosen variable (branching literal).
- Continue the algorithm recursively.
- **Backtracking**: If we arrive to a contradiction, we return to the last choice, and we "branch" by assigning **false** to the chosen variable.

The DPLL Algorithm (figure 2.1)

bool function Algo_DPLL(Γ: set of clauses)

0 Remove the valid clauses from Γ.
If $\Gamma = \emptyset$, return (`true`).
Else return (`DPLL`(Γ))

bool function `DPLL`(Γ: set of non-valid clauses)
The function returns true if and only if Γ is satisfiable.

1 **If** $\bot \in \Gamma$, return(`false`).
If $\Gamma = \emptyset$, return (`true`).

2 Reduce Γ.

3 Remove from Γ the clauses containing a pure literal.
If the set Γ has been modified, goto 1.

4 Apply unit resolution to Γ.
If the set Γ has been modified, goto 1.

5 Pick an arbitrary variable x in Γ
return (`DPLL`($\Gamma[x := 0]$) or else `DPLL`($\Gamma[x := 1]$))

Removal of clauses containing a pure literal

Definition 2.2.1

A litteral L is **pure** if none of the clauses in Γ contains L^c.

Lemma 2.2.2

Removing clauses with a pure literal preserves satisfiability.

Proof: see exercise 49.

Intuition: assigning $[L]_v$ to 1 is always possible for a pure literal.

Example 2.2.3

Let Γ be the set of clauses

(1) $p + q + r$

(2) $\overline{q} + \overline{r}$

(3) $q + s$

(4) $\overline{s} + t$

Simplify Γ by removing clauses containing pure literals.

p and t are pure.

(2) $\overline{q} + \overline{r}$

(3) $q + s$

$\overline{r}$ and s are now pure.
We obtain the empty set.
Therefore Γ has a model (for instance $p = 1, t = 1, r = 0, s = 1$).

Unit resolution

Definition 2.2.4

A unit clause is a clause which contains only one literal.

Let L be the literal from a unit clause of Γ.
Let Θ be the set of clauses obtained by:

- removing the clauses containing L
- removing L^c **inside** the remaining clauses
- if Γ contains two complementary unit clauses, then $\Theta = \{\bot\}$.

We apply this process for every unit clause.

Lemma 2.2.5

Γ has a model if and only if Θ has a model.

Proof: The proof is requested in exercise 50.

Example 2.2.6 Unit resolution

Simplify the following sets of clauses by unit resolution:

- $\Gamma = p + q,\ \bar{p},\ \bar{q}$

  $q,\ \bar{q}$ by unit resolution on $\bar{p}$,
  then $\bot$ by UR on $\bar{q}$
  Hence Γ has no model.

- $\Gamma = a + b + \bar{d},\ \bar{a} + c + \bar{d},\ \bar{b},\ d,\ \bar{c}$

  1. $a, \bar{a}$.
  2. $\bot$

  hence Γ has no model.

- $\Gamma = p,\ q,\ p + r,\ \bar{p} + r,\ q + \bar{r},\ \bar{q} + s$

  By unit resolution, we obtain: r, s.
  r, s has a model, hence Γ has one too.

The DPLL Algorithm (figure 2.1)

bool function Algo_DPLL(Γ: set of clauses)

0 Remove the valid clauses from Γ.
If $\Gamma = \emptyset$, return (`true`).
Else return (`DPLL`(Γ))

bool function `DPLL`(Γ: set of non-valid clauses)

The function returns true if and only if Γ is satisfiable.

1 **If** $\bot \in \Gamma$, return(`false`).
If $\Gamma = \emptyset$, return (`true`).

2 Reduce Γ.

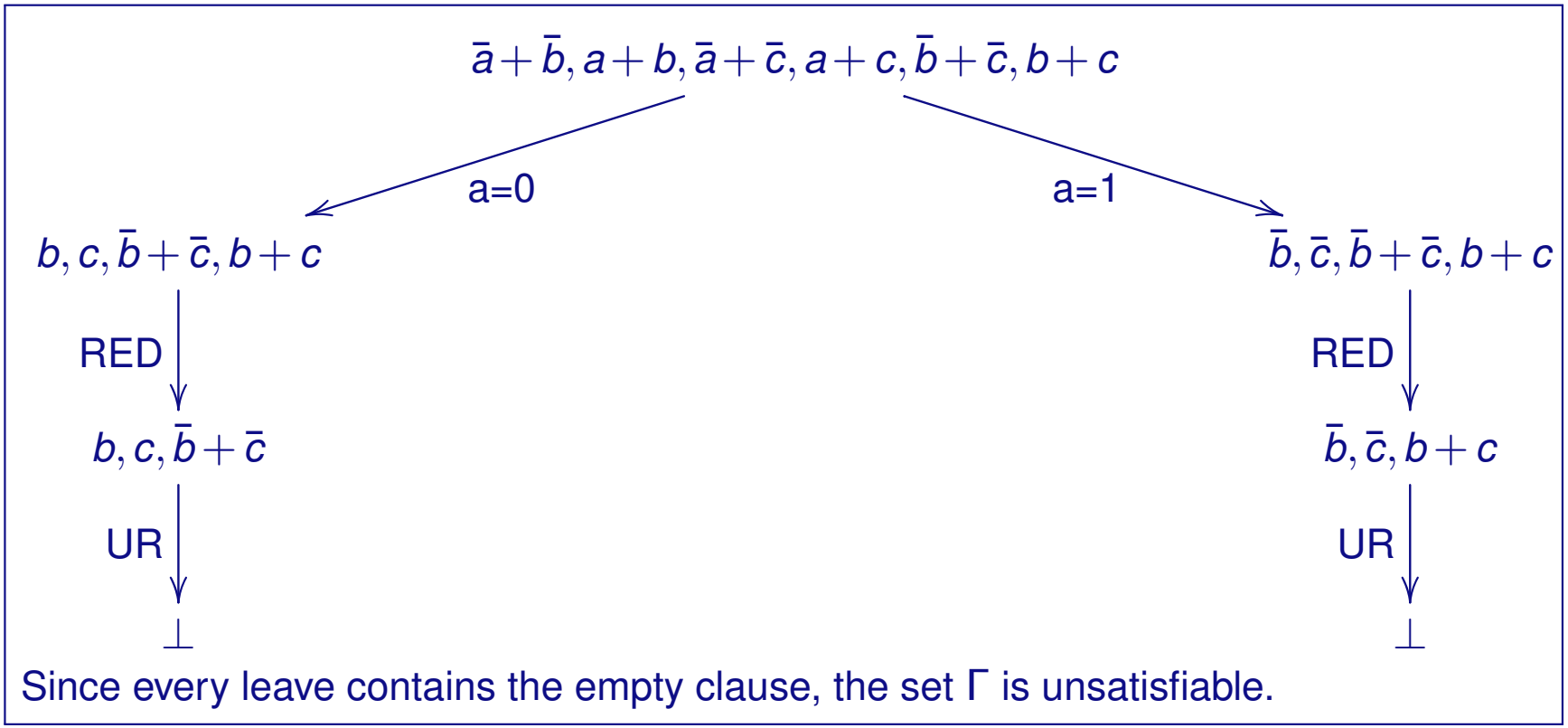
3 Remove from Γ the clauses containing a pure literal.
If the set Γ has been modified, goto 1.

4 Apply unit resolution to Γ.
If the set Γ has been modified, goto 1.

5 Pick an arbitrary variable x in Γ
return (`DPLL`($\Gamma[x := 0]$) or else `DPLL`($\Gamma[x := 1]$))

Example 2.2.8

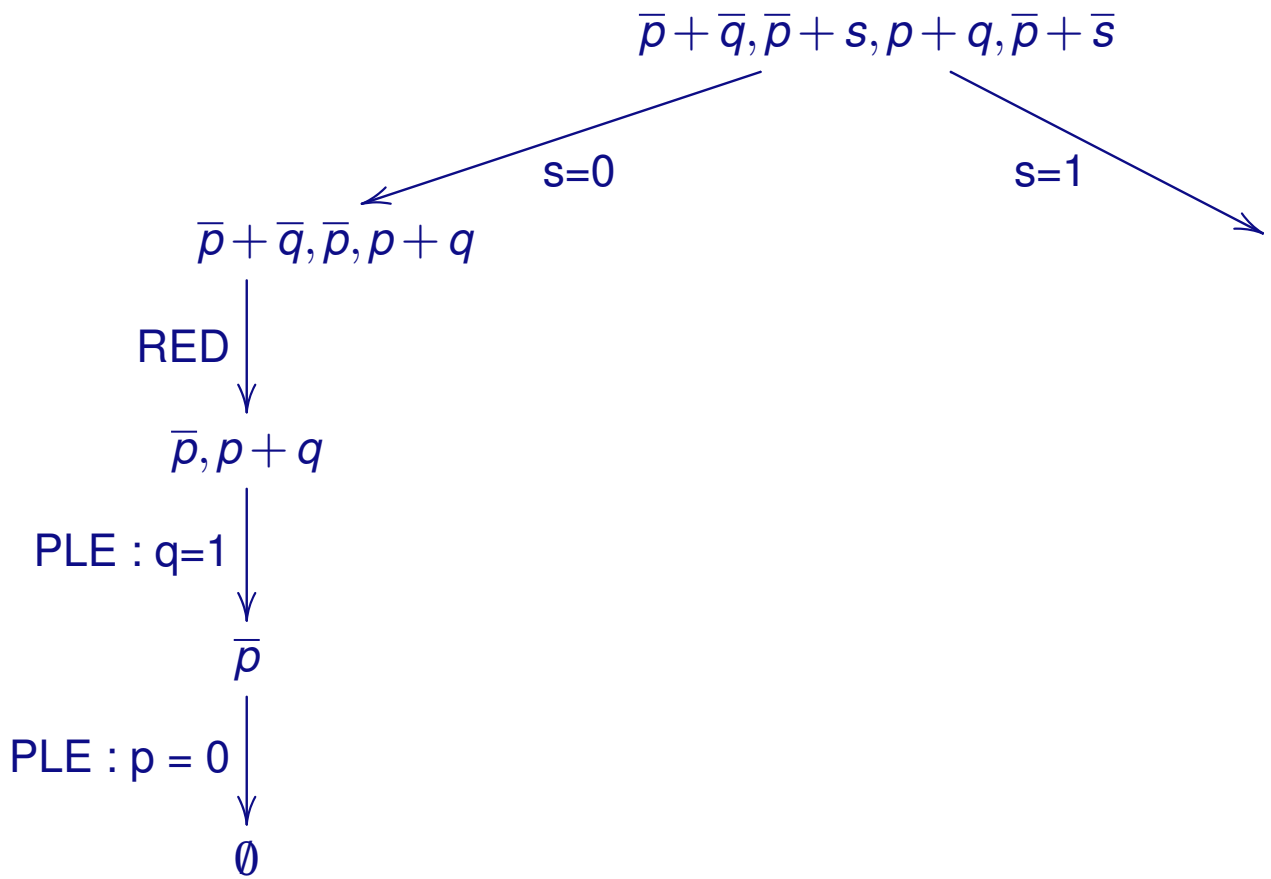
Let Γ be the set of clauses: $\overline{a}+\overline{b}$, $a+b$, $\overline{a}+\overline{c}$, $a+c$, $\overline{b}+\overline{c}$, $b+c$.



Since every leave contains the empty clause, the set Γ is unsatisfiable.

Example 2.2.8

Let Γ be the set of clauses: $\overline{p}+\overline{q}$, $\overline{p}+s$, $p+q$, $\overline{p}+\overline{s}$.



Since one branch leads to the empty set, the set Γ is satisfiable.
It is useless to continue the construction of the right branch.

Theorems 2.2.9 et 2.2.10

The algorithm Algo_DPLL is correct and terminates.

Termination proof

- Valid clause removal is only executed once
- Simplification iteration: the number of clauses strictly decreases
- Recursive calls: the number of variables strictly decreases

Hence the termination.

Correctness proof

The point is to prove that satisfiability is preserved through the whole algorithm.

- In the main loop:
  see lemma for each simplification.
- Correctness of recursive calls:
  Reminder of property 2.1.21:
  Γ *has a model iff* $\Gamma[x := 0]$ *or* $\Gamma[x := 1]$ *is satisfiable.*
  So if the recursive calls are correct, the current call is too.

Since the algorithm is correct for a set Γ with no literal (base cases), it is correct for any set Γ of clauses.

Remarks 2.2.11 and 2.2.12

- **Forgetting simplifications:** DPLL is still correct if we forget (once or more) reduction (2), pure literal elimination (3) and/or unit reduction (4).
- **Choice of the variable (branching literal):**
  - A good choice for variable x in step (5) is the variable that appears most often.
  - A better choice is the variable which will lead to the maximum number of simplifications

Cf. Sub-section 2.2.5, for the main branching heuristics

Principle of the algorithm: Build all the clauses deduced from Γ

Following the height of the proof trees.

Algorithm

For any integer i
While it is possible to construct new clauses
Build the reduced set of all the clauses having a proof tree of height at most i.

In practice:
Maintain two sequences of the sets of clauses, $\Delta_{i(i\geq 0)}$ and $\Theta_{i(i\geq 0)}$

Two sequences of sets of clauses

Δ_i are the new useful clauses

Clauses deduced from Γ by a proof of height i, after removal of:

- valid clauses
- clauses including another clause whose proof has height $< i$.

Δ_0 is obtained by reducing Γ.

Θ_i are the old clauses still useful

Clauses deduced from Γ by a proof of height $< i$ after removal of:

- valid clauses
- clauses including another clause whose proof has height $\leq i$.

Θ_0 is the empty set.

Exemple ??

Soit $\Gamma = \{a+b+\bar{a}, a+b, a+b+c, a+\bar{b}, \bar{a}+b, \bar{a}+\bar{b}\}$

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Résolvants de Δ_i et $\Delta_i \cup \Theta_i$
0	~~$a+b+\bar{a}$~~, $a+b$ ~~$a+b+c$~~, $a+\bar{b}$ $\bar{a}+b, \bar{a}+\bar{b}$	$\emptyset$	~~$a+b$~~, ~~$a+\bar{b}$~~, ~~$\bar{a}+b$~~, ~~$\bar{a}+\bar{b}$~~	a, b, ~~$b+\bar{b}$~~, ~~$a+\bar{a}$~~, $\bar{b}, \bar{a}$
1	$a, b, \bar{b}, \bar{a}$	$\emptyset$	$a, b, \bar{b}, \bar{a}$	$\bot$
2	$\bot$	$\emptyset$	$\bot$	$\emptyset$
3	$\emptyset$	$\bot$		

Reminder:

- $\Delta_{i+1} =$
  - Compute all the resolvents of Δ_i and $\Delta_i \cup \Theta_i$
  - Reduce this set
  - Remove the new resolvents which include a clause from $\Delta_i \cup \Theta_i$
- $\Theta_{i+1} =$
  Remove from $\Delta_i \cup \Theta_i$ the clauses which include a clause of Δ_{i+1}.

Today

- Resolution is a correct and complete deductive system: it **characterizes** all the unsatisfiable formulas.
- The DPLL **algorithm** uses ideas from resolution to:
  - find a **model**
  - otherwise, prove the **unsatisfiability** by an efficient search of the assignments.
- Complete Strategy is an **algorithm** for computing **every** clause deducible from an initial set

Next lecture

- Natural deduction

Homework: **Hypotheses** :

- (H1) : $p \Rightarrow \neg j \equiv \bar{p} + \bar{j}$
- (H2) : $\neg p \Rightarrow j \equiv p + j$
- (H3) : $j \Rightarrow m \equiv \bar{j} + m$
- ($\neg$ C): $\neg m \wedge \neg p$ (two clauses $\bar{m}$ and $\bar{p}$)

Build the proof of $H1, H2, H3, \neg C \vdash \bot$ obtained by the DPLL algorithm (you may pick any variable for branching)